Model-Checking Games for Typed lambda-Calculi

Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 172 (2007) 589–609
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Model-Checking Games for Typed λ-Calculi

Colin Stirling[1]

School of Informatics
Edinburgh University
Edinburgh, UK

Abstract

We consider the transfer of verification techniques to structures with binding.

Keywords: games, typed lambda calculus, higher-order matching, higher-order schemes.

1 Introduction

A notable success in Computer Science has been the development of techniques for the computer-assisted verification of finite and infinite-state systems. These methods include model checking and equivalence checking. A general research goal is to transfer them to classes of finite and infinite-state systems with binding. In this paper we examine two problems involving typed λ-calculi, *higher-order matching* and *higher-order program schemes*.

The basic idea for understanding both these problems is to view typed λ-terms as *models* with binding, akin to transition graphs, and to understand their dynamical behaviour (β-reduction) by playing games on them without changing them using substitution. In the case of matching we assume we are given a potential solution term t. The model-checking game for t decides whether it is a solution. To transform this into a decision procedure for matching, one needs to find play unformities that imply the small model property: if a problem has a solution then it has a small solution [26]. Here, we present the proof for the third-order case, as the small model property follows immediately from the tree-model property that every play descends a branch of a solution term. In the case of a scheme, we adopt Ong's presentation as an infinite term (that can be wrapped into a finite graph) [17]. We

[1] Email: cps@inf.ed.ac.uk

doi:10.1016/j.entcs.2007.02.021

define a scheme model checking game similar in spirit to game semantics as defined by Ong [17] but based on the ideas introduced for defining matching games.

In Section 2 we introduce higher-order matching and its equivalent problem, dual interpolation. The matching game is defined in Section 3 and in Section 4 we show decidability for the 3rd-order case. Higher-order schemes are introduced in Section 5 and their games in Section 6.

2 Matching and dual interpolation

We consider simple types that are generated from a single base type $\mathbf{0}$ using the binary $\rightarrow$ operator. A type is $\mathbf{0}$ or $A \rightarrow B$ where A and B are types. If $A \neq \mathbf{0}$ then it has the form $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow \mathbf{0}$, assuming $\rightarrow$ associates to the right, which is here written $(A_1, \ldots, A_n, \mathbf{0})$ following Ong [17]. A standard definition of *order* is: the order of $\mathbf{0}$ is 1 and the order of $(A_1, \ldots, A_n, \mathbf{0})$ is $k+1$ where k is the maximum of the orders of the A_is.

Terms of the simply typed λ-calculus are built from a countable set of typed variables $x, y, \ldots$ and typed constants $a, f, \ldots$ (each variable and constant has a unique type). The set of simply typed terms is the smallest set T such that

(i) if x (f) has type A then $x : A \in T$ ($f : A \in T$),

(ii) if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \rightarrow B \in T$,

(iii) if $t : A \rightarrow B \in T$ and $u : A \in T$ then $tu : B \in T$.

The *order* of a typed term is the order of its type. A typed term is *closed* if it does not contain free variables. Throughout, we assume the definitions of α-equivalence, β and η-reduction.

Definition 2.1 A *matching problem* is an equation $v = u$ where $v, u : \mathbf{0}$ and u is closed. The *order* of the problem is the maximum of the orders of the free variables $x_1, \ldots, x_n$ in v. A *solution* is a sequence of terms $t_1, \ldots, t_n$ such that $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u$.

The decision question is: given a matching problem, does it have a solution? Matching is a particular instance of higher-order unification when the term u is closed: can v be pattern matched to u? Although higher-order unification is undecidable (even if free variables are only second-order), higher-order matching was conjectured to be decidable by Huet [12]. If matching is decidable then it is known to have non-elementary complexity. Decidability has been proved for the general problem up to order 4 using observational equivalence of λ-terms and for various special cases [18,19]. Comon and Jurski define tree automata that characterize all solutions to a 4th-order problem, thereby, showing that they form a regular set [6]. Loader showed that matching is undecidable for the variant definition when β-equality is the same normal form by encoding λ-definability as matching [15], also see [13].

In [26], we describe a procedure that shows that matching is decidable that uses finite model checking games on λ-terms. In this paper, we describe the model checking game and how it leads to decidability for the third-order case, as a prelude

to examining (infinite) model checking games for higher-order schemes.

We assume that all terms in *normal form* are in η-*long form*,

(i) if $t : \mathbf{0}$ then it is $u : \mathbf{0}$ where u is a constant or a variable, or $u\, t_1 \dots t_k$ where $u : (B_1, \dots, B_k, \mathbf{0})$ is a constant or a variable and each $t_i : B_i$ is in η-long form,

(ii) if $t : (A_1, \dots, A_n, \mathbf{0})$ then t is $\lambda y_1 \dots y_n.t'$ where each $y_i : A_i$ and $t' : \mathbf{0}$ is in η-long form.

Throughout we use $\lambda z_1 \dots z_n$ for $\lambda z_1 \dots \lambda z_n$. A term is *well-named* if each occurrence of a variable y within a λ-abstraction is unique.

Assume $u : \mathbf{0}$ and each $v_i : A_i$, $1 \leq i \leq n$, is a closed term in normal form and $x : (A_1, \dots, A_n, \mathbf{0})$. An *interpolation equation* has the form $xv_1 \dots v_n = u$ and an *interpolation disequation* is $xv_1 \dots v_n \neq u$. A finite family of interpolation equations $xv_1^i \dots v_n^i = u_i$, $i : 1 \leq i \leq m$, with the same free variable x is an *interpolation problem* P. A finite family of interpolation equations and disequations, $xv_1^i \dots v_n^i \approx_i u_i$, $i : 1 \leq i \leq m$ and each $\approx_i \in \{=, \neq\}$, with the same free variable x is a *dual interpolation problem* P. The type of problem P is that of x and the order of P is the order of x. A *solution* of P of type A is a closed term $t : A$ in normal form such that for each equation $tv_1^i \dots v_n^i =_\beta u_i$ and for each disequation $tv_1^i \dots v_n^i \neq_\beta u_i$. We write $t \models P$ if t is a solution of P.

Schubert shows that a matching problem of order n reduces to an interpolation problem of order at most $n+2$ and Padovani shows it reduces to a dual interpolation problem of order n [20,19]. Because of normal forms, β-equality and $\beta\eta$-equality coincide. Consequently, the higher-order matching problem reduces to the following decision question: given a (dual) interpolation problem P, is there a term $t \models P$?

Example 2.2 The problem $x\, \lambda y_1 y_2.y_1 \lambda y_3.f y_3 y_3 = faa$ from [6] has order 3 with $x : ((\mathbf{0}, \mathbf{0}, \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0})$ and each $y_i : \mathbf{0}$. □

Example 2.3 The problem $x\, \lambda z.z = f(\lambda x_1 x_2 x_3.x_1(x_3))a$ also has order 3 where x has type $((\mathbf{0}, \mathbf{0}), \mathbf{0})$. This example illustrates that the closed term $u : \mathbf{0}$ may contain bound variables: here $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$. □

3 Matching games

We use a game-theoretic characterization of dual interpolation inspired by model checking games (such as in [23]) where a model, a transition graph, is traversed relative to a property and players make choices at appropriate positions. Similarly, in the following game the model is a putative solution term t that is traversed relative to the dual interpolation problem P. The central motivation is to model the dynamics, β-reduction, without changing t by substituting into it. Because of binding play may jump around t.

A potential solution term t for P has the right type, is in normal form, is well-named (with variables that are disjoint from variables in P). Term t is represented as a tree, tree(t). If t is $y : \mathbf{0}$ or $a : \mathbf{0}$ then tree(t) is the single node labelled with t. In the case of $uv_1 \dots v_k$ when u is a variable or a constant, we assume that a dummy

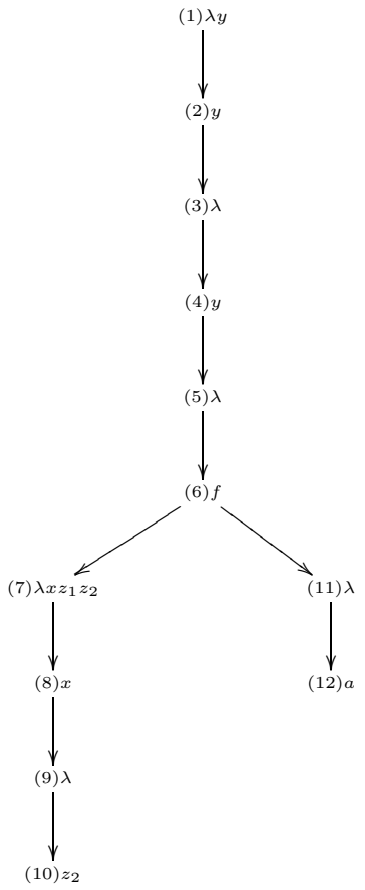


Fig. 1. A solution term

λ with the empty sequence of variables is placed before any subterm $v_i : \mathbf{0}$ in its tree representation. With this understanding, if t is $uv_1 \dots v_k$ then tree(t) consists of the root node labelled u and k-successor nodes labelled with tree(v_i). We use the notation $u \downarrow_i t'$ to represent that tree t' is the ith successor of the node u. If t is $\lambda\overline{y}.v$, where $\overline{y}$ is possibly empty, then tree(t) consists of the root node labelled $\lambda\overline{y}$ and a single successor node tree(v), $\lambda\overline{y} \downarrow_1$ tree(v).

In the following we use t to be the λ-term t, its λ-tree or the label (a constant, variable or $\lambda\overline{y}$) at its root node.

Example 3.1 A solution term t for the problem of Example 2.3 is $\lambda y.y(y(f(\lambda x z_1 z_2.x(z_2))\, a))$. The tree for t (without indices on the edges) is in Figure 1. For instance, in this tree (6) $\downarrow_2$ (11). There are various nodes in the tree with dummy λs such as (5) and (9). □

We assume that each node of a tree t is uniquely identifiable: for instance, in Example 3.1 we labelled each node with a distinct natural number to distinguish different occurrences of z and λ.

Innocent game semantics following Ong in [17] provides a possible game-theoretic foundation. Given a potential solution term t and a (dis)equation $xv_1^i \dots v_n^i \approx_i u_i$ there is the game board in Figure 2. Player Opponent chooses a branch of u_i.

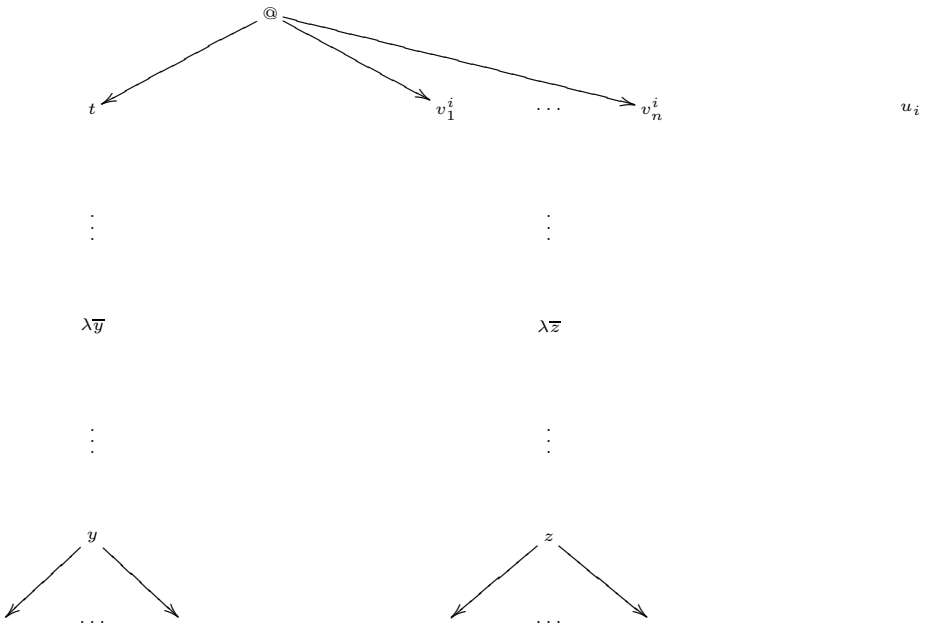


Fig. 2. Illustrating game semantics

Then, there is a finite play that starts at the root of t and may repeatedly jump in and out of t and in and out of the v_j^i's. At a constant $a : \mathbf{0}$ play ends. At other constants, player Proponent tries to match Opponent's choice of branch. Proponent wins, when the play finishes, if the sequence of constants encountered matches the branch chosen by Opponent. Play, for example, may reach y in t and then jump to $\lambda\overline{z}$ in v_j^i, as it is the subtree at $\lambda\overline{z}$ that is applied to $\lambda\overline{y}$, and then when at z in v_j^i play may return to t to an immediate successor of y. Game semantics models β-reduction on the fixed structure of Figure 2 without changing it using substitution. This is the rationale for the tree-checking game. However, it starts from the assumption that only t is the common structure for the problem P. So, play will always be in t. Jumping in and out of the v_j^i's is coded using states, as play traverses t. Moreover, the game avoids the justification pointers of game semantics by appealing to iteratively defined look-up tables.

The tree-checking game $\mathsf{G}(t, P)$ is played by one participant, player $\forall$, the *refuter* who attempts to show that t is not a solution of P. It appeals to a finite set of states involving *left terms*, subterms of the v_j^i's, and *right terms*, closed subterms of the u_i's, of the matching (dis)equations in P (both modulo substitution of constants for bound variables that are directly below a constant f, as we shall see). There are three kinds of state: *argument*, *value* and *final* states. An argument state has the form $q[(l_1, \ldots, l_k), r]$ where each l_j is a left term (and k can be 0) and r is a right term. Such a state will occur at a node labelled $\lambda z_1 \ldots z_k$ in t where each l_j has the same type as z_j: $(l_1, \ldots, l_k)$ are the subterms applied to $\lambda z_1 \ldots z_k$. A value state has the form $q[l, r]$ where l is a left term and r a right term. This state is associated with a node labelled with a variable y in t where y and l share the same type: l is the subterm of some v_j^i that play at y would jump to in game semantics. A final state is either $q[\,\forall\,]$ or $q[\,\exists\,]$.

A. $t_m = \lambda y_1 \dots y_j$, $t_m \downarrow_1 t'$ and $q_m = q[(l_1, \dots, l_j), r]$.
So, $t_{m+1} = t'$, $\theta_{m+1} = \theta_m\{l_1\eta_m/y_1, \dots, l_j\eta_m/y_j\}$ and q_{m+1}, η_{m+1} are defined by cases on t_{m+1}.
1. $a : \mathbf{0}$. So, $\eta_{m+1} = \eta_m$. If $r = a$ then $q_{m+1} = q[\exists]$ else $q_{m+1} = q[\forall]$.
2. $f : (B_1, \dots, B_k, \mathbf{0})$. So, $\eta_{m+1} = \eta_m$. If $r = fs_1 \dots s_k)$ then $q_{m+1} = q_m$ else $q_{m+1} = q[\forall]$.
3. $y : B$. If $\theta_{m+1}(y) = l\eta_i$, then $\eta_{m+1} = \eta_i$ and $q_{m+1} = q[l, r]$.

B. $t_m = f : (B_1, \dots, B_k, \mathbf{0})$, $t_m \downarrow_i t'_i$ and $q_m = q[(l_1, \dots, l_j), fs_1 \dots s_k]$.
So, $\theta_{m+1} = \theta_m$, $\eta_{m+1} = \eta_m$ and $\forall$ chooses a direction $d : 1 \leq d \leq k$.
1. $t_{m+1} = t'_d$. If $s_d : \mathbf{0}$, then $q_{m+1} = q[(\ ), s_d]$. If s_d is $\lambda x_1 \dots x_n.s$ then $q_{m+1} = q[(c_1, \dots, c_n), s\{c_1/x_1, \dots, c_n/x_n\}]$ where the c_i's are new constants and each c_i has the same type as x_i.

C. $t_m = y$ and $q_m = q[l, r]$.
If $l = \lambda z_1 \dots z_j.w$ and $t_m \downarrow_i t'_i$ then $\eta_{m+1} = \eta_m\{t'_1\theta_m/z_1, \dots, t'_j\theta_m/z_j\}$ else $\eta_{m+1} = \eta_m$. Elements t_{m+1}, q_{m+1} and θ_{m+1} are by cases on l.
1. $a : \mathbf{0}$ or $\lambda\overline{z}.a$. So, $t_{m+1} = t_m$ and $\theta_{m+1} = \theta_m$. If $r = a$ then $q_{m+1} = q[\exists]$ else $q_{m+1} = q[\forall]$.
2. $c : (B_1, \dots, B_k, \mathbf{0})$. So, $\theta_{m+1} = \theta_m$. If $r \neq cs_1 \dots s_k$ then $t_{m+1} = t_m$ and $q_{m+1} = q[\forall]$. Otherwise, $\forall$ chooses a direction $d : 1 \leq d \leq k$ and $t'_{m+1} = t'$ where $t'_m \downarrow_d t'$. If $s_d : \mathbf{0}$ then $q_{m+1} = q[(\ ), s_d]$. If s_d is $\lambda x_1 \dots x_n.s$ then $q_{m+1} = q[(c_1, \dots, c_n), s\{c_1/x_1, \dots, c_n/x_n\}]$ where the c_i's are new constants and each c_i has the same type as x_i.
3. $fw_1 \dots w_k$ or $\lambda\overline{z}.fw_1 \dots w_k$. So, $t_{m+1} = t_m$ and $\theta_{m+1} = \theta_m$. If $r \neq fs_1 \dots s_k$ then $q_{m+1} = q[\forall]$. Otherwise, $\forall$ chooses a direction $d : 1 \leq d \leq k$. If $s_d : \mathbf{0}$ then $q_{m+1} = q[w_d, s_d]$. If $s_d = \lambda x_1 \dots x_n.s$ and $w_d = \lambda y_1 \dots y_n.w$ then $q_{m+1} = q[w\{c_1/y_1, \dots, c_n/y_n\}, s\{c_1/x_1, \dots, c_n/x_n\}]$ where the c_i's are new constants and each c_i has the same type as x_i and y_i.
4. $xl_1 \dots l_k$ or $\lambda\overline{z}.xl_1 \dots l_k$. If $\eta_{m+1}(x) = t'\theta_i$ then $t_{m+1} = t'$, $\theta_{m+1} = \theta_i$ and $q_{m+1} = q[(l_1, \dots, l_k), r]$.

Fig. 3. Game moves

As play proceeds in t, there are two kinds of free variable: those in t, such as y in Figure 2, and those in the left terms of states, such as z in Figure 2. Free variables in t are associated with left terms and free variables in states are associated with nodes of t. So, the game appeals to a sequence of supplementary look-up tables θ_k and η_k, $k \geq 1$: θ_k is a partial map from variables in t to pairs $l\eta_j$ where l is a left term and $j < k$, and η_k is a partial map from free variables in subterms of $v^i_{j'}$ to pairs $t'\theta_j$ where t' is a node of the tree t and $j < k$. A variable y in t may be associated with a left subterm l which contains free variables: hence, the need for $\theta_k(y)$ to be a pair $l\eta_j$ as η_j records the values of the free variables in l and $j < k$. Similarly, a variable in a left subterm may be associated with a subtree of t which contains free variables. Initially, at the beginning of play when there are no free variables, θ_1 and η_1 are both empty.

(1) $q[(\lambda z.z), u]\,\theta_1\,\eta_1$			
(2) $q[\lambda z.z, u]\,\theta_2\eta_2$	$\theta_2 = \theta_1\{(\lambda z.z)\eta_1/y\}$	$\eta_2 = \eta_1$	$A3$
(3) $q[(\ ), u]\,\theta_3\eta_3$	$\theta_3 = \theta_2$	$\eta_3 = \eta_2\{(3)\theta_2/z\}$	$C4$
(4) $q[\lambda z.z, u]\,\theta_4\eta_4$	$\theta_4 = \theta_3$	$\eta_4 = \eta_1$	$A3$
(5) $q[(\ ), u]\,\theta_5\,\eta_5$	$\theta_5 = \theta_4$	$\eta_5 = \eta_4\{(5)\theta_4/z\}$	$C4$
(6) $q[(\ ), u]\,\theta_6\,\eta_6$	$\theta_6 = \theta_5$	$\eta_6 = \eta_5$	$A2$

Fig. 4. Initial moves

A *play* of $\mathsf{G}(t, P)$ is a sequence of positions $t_1q_1\theta_1\eta_1, \ldots, t_nq_n\theta_n\eta_n$ where each t_i is (the label of) a node of t, $t_1 = \lambda\overline{y}$ is the root node of t, each q_i is a state and q_n is a final state. A node t' of the tree t may repeatedly occur in a play. The initial state is decided as follows: $\forall$ chooses a (dis)equation $x(v_1^i, \ldots, v_n^i) \approx_i u_i$ from P and $q_1 = q[(v_1^i, \ldots, v_n^i), u_i]$. This is the same as an initial position in the game semantics, except the terms v_j^i and u_i are now part of the state (and the choice of branch in u_i takes place as play proceeds).

If the current position is $t_mq_m\theta_m\eta_m$ and q_m is not a final state, then the next position $t_{m+1}q_{m+1}\theta_{m+1}\eta_{m+1}$ is determined by a unique move in Figure 3. Moves are divided into three groups that depend on t_m. Group A covers the case when it is a $\lambda\overline{y}$, group B when it is a constant f (whose type is not $\mathbf{0}$) and group C when it is a variable y. We assume standard updating notation for θ_{m+1} and η_{m+1}: $\beta\{\alpha_1/y_1, \ldots, \alpha_m/y_m\}$ is the partial function similar to β except that $\beta(y_i) = \alpha_i$. In the case of rules B1, C2 and C3 we assume that the constants c_i are new: their role is to replace bound variables directly beneath a constant f. These are also the only rules where $\forall$ can exercise choice, by carving out a branch. The look-up tables are used in rules A3 and C4 to interpret the two kinds of free variables. If t_m is a λ node, $t_m \downarrow_1 t_{m+1}$ and t_{m+1} is the variable y, then η_{m+1} and q_{m+1} are determined by the entry for y in θ_{m+1}: if it is $l\eta_i$, then l is the left element of q_{m+1} and $\eta_{m+1} = \eta_i$. In C4, if $t_m = y$, $q_m = q[l, r]$ and $l = x(l_1, \ldots, l_k)$ or $\lambda\overline{z}.x(l_1, \ldots, l_k)$, then θ_{m+1} and t_{m+1} are determined by the entry for x in the table η_{m+1}: if it is $t'\theta_i$ then $t_{m+1} = t'$ and $\theta_{m+1} = \theta_i$. It is this rule that allows play to jump elsewhere in the term tree (always to a node labelled with a λ). In contrast, for A1-A3, B1 and C2 (unless play finishes) control passes down the term tree while it remains stationary in the case of C1 and C3.

A play of $\mathsf{G}(t, P)$ finishes with a final state $q[\,\forall\,]$ or $q[\,\exists\,]$. Player $\forall$ *wins* the play if the final state is $q[\,\forall\,]$ and she *loses* it if the final state is $q[\,\exists\,]$.

Example 3.2 Let P be the problem of Example 2.3 and let t be the term tree of Figure 1. $\mathsf{G}(t, p)$ consists of two plays that descend t. Both plays start as in Figure 4 where we have supplied which move is applied to produce the next position. The initial state is an argument state $q[(\lambda z.z), u]$ and control is at node (1). Play descends from node (1) to (2) calling the value $\lambda z.z$ by A3. Next, by C4, because z is the head variable in the body of $\lambda z.z$, has no arguments and is associated with node (3), the next state is the argument state $q[(\ ), u]$ and control is at (3). Play descends from (3) to (4) calling the value $\lambda z.z$ by A3. Again by C4, z is the head variable in the body of $\lambda z.z$, has no arguments and is now associated with (5), the

next state is the argument state $q[(\ ), u]$ and control is at (5). A2 is then applied (because the right term u in the state has f as head constant) and control passes from (5) to (6). Move B1 is now applied and there is a $\forall$ choice as to which branch of u to take. If direction 1 is chosen then play continues as follows.

$(7)\ q[(c_1, c_2, c_3), c_1(c_3)]\ \theta_7\ \eta_7$	$\theta_7 = \theta_6$	$\eta_7 = \eta_6$	$B1$
$(8)\ q[c_1, c_1(c_3)]\ \theta_8\ \eta_8$	$\theta_8 = \theta_7\{c_1\eta_7/x, c_3\eta_7/z_2\}$	$\eta_8 = \eta_7$	$A3$
$(9)\ q[(\ ), c_3]\ \theta_9\ \eta_9$	$\theta_9 = \theta_8$	$\eta_9 = \eta_8$	$C2$
$(10)\ q[c_3, c_3]\ \theta_{10}\ \eta_{10}$	$\theta_{10} = \theta_9$	$\eta_{10} = \eta_7$	$A3$
$(10)\ q[\ \exists\ ]\ \theta_{11}\ \eta_{11}$	$\theta_{11} = \theta_{10}$	$\eta_{11} = \eta_{10}$	$C1$

New constants are introduced for replacing x_1, x_2, x_3 in the body of u to give the right term $c_1(c_3)$ and as arguments (c_1, c_2, c_3) for the variables x, z_1, z_2 bound at (7). At (8) the value c_1 is called and by C2, control proceeds to (9) where the right term is reduced to c_3. Finally, at (10), the value c_3 is called, and by C1, $\forall$ loses the play. She also loses if direction 2 is chosen. □

Example 3.3 A solution to Example 2.2, $x\, v_1 v_2 = faa$, is depicted in Figure 5 where $v_1 = \lambda y_1 y_2.y_1$ and $v_2 = \lambda y_3.fy_3y_3$. The moves are also shown. In fact there are two plays here: at position 4 there is a $\forall$-choice: however, both choices lead to the same position 5. □

If P is an interpolation problem then $\forall$ *loses the game* $\mathsf{G}(t, P)$ if she loses every play: for each equation she loses every play whose initial state is from it. When P is a dual interpolation problem, $\forall$ *loses the game* $\mathsf{G}(t, P)$ if for each equation she loses every play whose intial state is from it and if for each disequation she wins at least one play whose initial state is from it. The game *characterizes* dual interpolation and is proved in [25].

Proposition 3.4 $\forall$ *loses* $\mathsf{G}(t, P)$ *if, and only if,* $t \models P$.

The number of different plays in $\mathsf{G}(t, P)$ is bounded by the sum of the number of branches in the right terms u_i of P. We let π range over *subplays* that are consecutive subsequences of positions of any play of $\mathsf{G}(t, P)$.

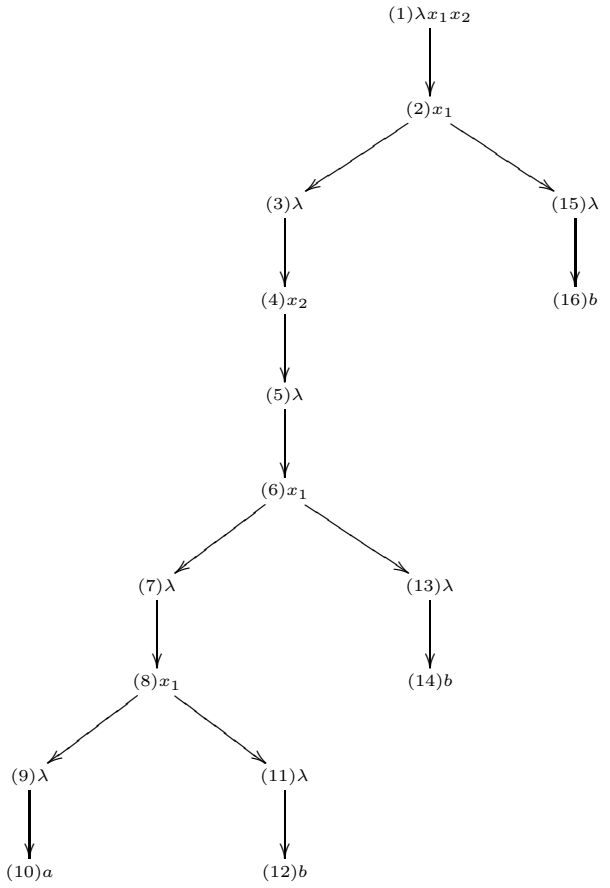
Definition 3.5 The *length* of π, $|\pi|$, is the number of positions in π. The *ith position* of π, for $1 \leq i \leq |\pi|$, is $\pi(i)$ and $\pi(i, j)$, $i \leq j$, is the *interval* $\pi(i), \ldots, \pi(j)$.

For ease of notation, we write $t \in \pi(i)$, $q \in \pi(i)$, $\theta \in \pi(i)$ and $\eta \in \pi(i)$ if $\pi(i) = tq\theta\eta$ and $t \notin \pi(i)$ means that $\pi(i) = t'q\theta\eta$ and $t \neq t'$. If $q = q[(l_1, \ldots, l_k), r]$ or $q[l, r]$ then its *right term* is r.

Definition 3.6 A subplay π is *ri*, *right term invariant*, if $q \in \pi(1)$ and $q' \in \pi(|\pi|)$ share the same right term r. It is *nri* if it is not ri and $q' \in \pi(|\pi|)$ is not final.

Definition 3.7 If $\pi \in \mathsf{G}(t, P)$ and $\pi(i)$'s look-up table is called when move A3 or C4 produces $\pi(j)$, $j > i$, then position $\pi(j)$ is a *child* of position $\pi(i)$. If $\pi(i+1)$ is the result of move B1 or C2, then $\pi(i+1)$ is a *child* of $\pi(i)$.

Fact 3.8 *If* $\pi \in \mathsf{G}(t, P)$, $j > 1$, $\pi(j)$ *is not a final position and* $\lambda\overline{y}$ *or* $y \in \pi(j)$, *then there is a unique* $\pi(i)$, $i < j$, *such that* $\pi(j)$ *is a child of* $\pi(i)$.



(1) $q[(v_1, v_2), faa]$	θ_1	η_1	
(2) $q[v_1, faa]$	$\theta_2 = \theta_1\{v_1\eta_1/x_1, v_2\eta_1/x_2\}$	$\eta_2 = \eta_1$	$A3$
(3) $q[(\,), faa]$	$\theta_3 = \theta_2$	$\eta_3 = \eta_2\{(3)\theta_2/y_1, (15)\theta_2/y_2\}$	$C4$
(4) $q[v_2, faa]$	$\theta_4 = \theta_3$	$\eta_4 = \eta_1$	$A3$
(4) $q[y_3, a]$	$\theta_5 = \theta_4$	$\eta_5 = \eta_4\{(5)\theta_4/y_3\}$	$C3$
(5) $q[(\,), a]$	$\theta_6 = \theta_4$	$\eta_6 = \eta_5$	$C4$
(6) $q[v_1, a]$	$\theta_7 = \theta_6$	$\eta_7 = \eta_1$	$A3$
(7) $q[(\,), a]$	$\theta_8 = \theta_7$	$\eta_8 = \eta_7\{(7)\theta_7/y_1, (13)\theta_7/y_2\}$	$C4$
(8) $q[v_1, a]$	$\theta_9 = \theta_8$	$\eta_9 = \eta_1$	$A3$
(9) $q[(\,), a]$	$\theta_{10} = \theta_9$	$\eta_{10} = \eta_9\{(9)\theta_9/y_1, (11)\theta_9/y_2\}$	$C4$
(10) $q[\,\exists\,]$	$\theta_{11} = \theta_{10}$	$\eta_{11} = \eta_{10}$	$A1$

Fig. 5. Plays of a solution term to Example 2.2

Definition 3.9 A look-up table β' *extends* β if for all $x \in \mathrm{dom}(\beta)$, $\beta'(x) = \beta(x)$.

Fact 3.10 *If $\pi(j)$ is a child of $\pi(i)$ then $\theta_j \in \pi(j)$ extends $\theta_i \in \pi(i)$ and $\eta_j \in \pi(j)$ extends $\eta_i \in \pi(i)$.*

4 Deciding third-order matching

We describe how the game-theoretic characterization of matching, Proposition 3.4, underpins decidability of third-order matching. The idea is to show the *small model property*: if $t_0 \models P$ then there is a *small* term $t' \models P$. First, we relate the static

structure of a solution tree t_0 with the dynamics of playing.

Definition 4.1 Assume $B = (B_1, \ldots, B_k, \mathbf{0})$.

(i) λ is an *atomic leaf* of type $\mathbf{0}$.

(ii) If $x_j : B_j$, $1 \leq j \leq k$, then $\lambda x_1 \ldots x_k$ is an *atomic leaf* of type B.

(iii) If $u : \mathbf{0}$ is a constant or variable then u is a *simple* tile.

(iv) If $u : B$ is a constant or a variable and $t_j : B_j$, $1 \leq j \leq k$, are atomic leaves then $u(t_1, \ldots, t_k)$ is a *simple* tile.

The term tree t_0 without its very top $\lambda\overline{y}$ consists of simple tile occurrences. In Figure 5, nodes (2),(3) and (15) form the simple tile $x_1(\lambda, \lambda)$ with atomic leaves λ and λ, (8), (9) and (11) form $x_1(\lambda, \lambda)$. Nodes such as (10) and (12) are also simple tiles without atomic leaves. The node (2) by itself or (2) with (3) are *not* simple tiles.

Throughout, our use of tile in t_0 means "tile occurrence" in t_0. We write $t(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ if t is a simple tile with atomic leaves $\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k$.

Definition 4.2 Assume t and t' are simple tiles.

(i) t' is *j-below* $t(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ in t_0 if there is a branch in t_0 from $\lambda\overline{x}_j$ to t'.

(ii) t' is an *immediate j-dependent* of tile t in t_0 if t' is j-below t and the head variable of t' is bound by a $\lambda\overline{y}$ in t.

(iii) t' is a *j-dependent* of t if it is an immediate j-dependent of t or there is a t'' that is an immediate j-dependent of t and t' is a j'-dependent of t'' for some j'.

(iv) t' is a *dependent* of t if it is a j-dependent of t for some j.

In Figure 1, the tile $x(\lambda)$ rooted at (8) is 1-below $f(\lambda x z_1 z_2, \lambda)$ rooted at (6) and is, therefore, also an immediate 1-dependent of it

Definition 4.3 Assume $t = t(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ is a simple tile in t_0.

(i) t is a *top* tile in t_0 if its free variable y is bound by the initial lambda $\lambda\overline{y}$ of t_0.

(ii) t is *j-end* in t_0, if every free variable below $\lambda\overline{x}_j$ in t_0 is bound above t. It is an *end* tile in t_0 if it is j-end for all $j : 1 \leq j \leq k$.

(iii) t is a *constant* tile in t_0 if its head is a constant or it is a dependent of a constant tile.

The tiles $x_1(\lambda, \lambda)$ rooted at (2), (6) and (8) are top and end tiles in Figure 5. Tiles $f(\lambda x z_1 z_2, \lambda)$ and $x(\lambda)$ in Figure 1 are constant tiles.

Tiles can also be categorized in terms of their dynamic properties by appealing to *subplays* of $\mathsf{G}(t_0, P)$.

Definition 4.4 A subplay π is a *play* on the simple tile $u(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ in t_0 if $u \in \pi(1)$, $\lambda\overline{x}_i \in \pi(|\pi|)$ for some i and $\pi(|\pi|)$ is a child of $\pi(1)$. It is a *j-play* if $\lambda\overline{x}_j \in \pi(|\pi|)$.

A play on a simple constant tile $u(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ is a pair of positions $\pi(i, i+1)$ with $u \in \pi(i)$ and $\lambda\overline{x}_j \in \pi(i+1)$ for some j (by moves B1 or C2 of Figure 3). If π is the play in Example 3.2, then $\pi(6,7)$ is a 1-play on $f(\lambda x z_1 z_2, \lambda)$ of Figure 1.

Fact 4.5 *If π is a play on a simple constant tile then $|\pi| = 2$.*

In the general higher-order matching case, a play π on a simple non-constant tile $y(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ in t_0 can be of arbitrary length. It starts at y and finishes at a leaf $\lambda\overline{x}_j$. In between, flow of control can be almost anywhere in t_0 (including y). However, because of paucity of binding, play is very restricted in the third-order case. In a play $\pi \in \mathsf{G}(t_0, P)$, it is not possible for there to be more than one subplay on a simple tile of t_0: in [26], it is shown that if $\pi(i,m)$ and $\pi(i,n)$, $n > m$, are plays on the simple tile $y(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ and $\lambda\overline{x}_j \in \pi(m)$ then there is a position $\pi(m')$, $m' < n$, that is a child of $\pi(m)$. Therefore, the following is an immediate corollary because a top tile in a third-order term tree has no dependent tiles.

Proposition 4.6 *If $\pi \in \mathsf{G}(t_0, P)$, P is third-order and t is a simple tile in t_0 then there is at most one subplay on t within π.*

Assume P is a 3rd-order problem and $t_0 \models P$. If we inspect t_0 top-down, starting beneath the initial λ, then it is a tree of simple tiles: each is a constant or a top tile that is also an end tile. The tree in Figure 1 consists of two initial top tiles $y(\lambda)$ that are also end tiles and the constant tiles $f(\lambda x z_1 z_2, \lambda)$, $x(\lambda)$, z_2 and a. Assume that $\Pi = \{\pi_1, \ldots, \pi_p\}$ are the plays of $\mathsf{G}(t_0, P)$. We define a partition of each $\pi \in \Pi$ in stages. At each stage n we examine a simple tile t_n in t_0 and a position $\pi(i_n)$ whose control is at the head of t_n. We formalize that the subplay $\pi(i_n, j_n)$ is a play on t_n or $j_n = |\pi|$. With each $\pi \in \Pi$ we associate a unique *colour* $c(\pi)$.

For stage 1, we identify the initial simple tile $t_1 = u(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ in t_0 which is a constant or top tile (and possibly $k = 0$). We examine the play on t_1, if there is one, consisting of moves $\pi(i_1, j_1)$ where $i_1 = 2$. If there is no play we let $j_1 = |\pi|$, so $q \in \pi(j_1)$ is final, and for all $i : i_1 \leq i \leq j_1$ it follows that $u \in \pi(i)$: so, control never reaches beyond this initial tile. Tile t_1 is then *final* for π and we terminate at this stage. Otherwise, the play on t_1 ends at one of its atomic leaves $\lambda\overline{x}_j$ and t_2 is the simple tile directly below $\lambda\overline{x}_j$ in t_0 and $i_2 = j_1 + 1$. If the play $\pi(i_1, j_1)$ on t_1 is nri then t_1 is coloured $c(\pi)$. At stage n, for any subsequent simple tile t_n we assume that the play $\pi(i_n, j_n)$ is the play on t_n, if there is one. If there is not a play then $j_n = |\pi|$ and t_n is final for π. If $\pi(i_n, j_n)$ is an nri play on t_n then t_n is coloured $c(\pi)$. In this way, the partition of π descends a branch of t_0 until it reaches a final tile.

Example 4.7 For the tree of Figure 1 and the play π of Example 3.2 there is the following partition of π.

$y \quad \lambda$	$y \quad \lambda$	$f \quad \lambda x z_1 z_2$	$x \quad \lambda$	z_2
$\pi(2,3)$	$\pi(4,5)$	$\pi(6,7)$	$\pi(8,9)$	$\pi(10,10)$

Tiles $f(\lambda x z_1 z_2, \lambda)$ and $x(\lambda)$ are coloured $c(\pi)$ and z_2 is final for π. For the other

play π' in this example, there is a similar partition.

$$\begin{array}{cccccccc} y & \lambda & y & \lambda & f & \lambda & a \\ \pi'(2,3) & & \pi'(4,5) & & \pi'(6,7) & & \pi'(8,8) \end{array}$$

where only $f(\lambda x z_1 z_2, \lambda)$ is coloured $c(\pi')$ and the tile a is final for π'. □

Consider partitioning with respect to all plays $\pi \in \Pi$. We slightly abuse notation: if π and π' are two plays we let $\pi(i_n, j_n)$, $\pi'(i_n, j_n)$ be subplays at stage n of their partition even when these intervals have different ranges. Instead of a sequence of simple tiles there is a tree of simple tiles, as they all share the initial tile. We select three kinds of simple tile in the tree. Tile t is *coloured* if it has at least one colour and t is *final* if it is final for at least one play. Each play at stage 1 that ends at the same atomic leaf of t_1 shares t_2 at stage 2 and so on. Therefore, branching occurs at t_m at stage m if there are plays that ended at the same atomic leaves of t_k, $k < m$, and at stage m plays end at different atomic leaves of t_m: tile t_m is then a (play) *separator*. In Example 4.7, $f(\lambda x z_1 z_2, \lambda)$ separates the plays. If a simple tile in t_0 is coloured, final or a separator then it is *special*.

A simple tile in t_0 with atomic leaves that is not special is superfluous. Either every play avoids it or every subplay that passes through it is ri and ends at the same atomic leaf $\lambda\overline{x}_j$. If every play avoids the simple tile $u(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ then we can replace the subtree rooted at u in t_0 with the simple constant tile $b : \mathbf{0}$. If every subplay that passes through $u(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ is ri and ends at the same atomic leaf $\lambda\overline{x}_j$ and t_j is the subtree beneath $\lambda\overline{x}_j$ of t_0, then we can replace the subtree rooted at u in t_0 by t_j. Clearly, both these transformations preserve solution trees. For instance, the two occurrences of $y(\lambda)$ in Example 4.7 are both redundant: the tree in Figure 1 is transformed by moving node (6) directly beneath node (1). The only significant tiles with atomic leaves are special. However, the number of coloured and final tiles is bound by the sum of the depths of the right terms u_i and the number of separators is at most $p-1$ (where p is the number of plays). So, the small model property follows where $s(P)$ is the appropriate measure.

Fact 4.8 *If t_0 is a smallest solution to third-order P, then $|t_0| \leq s(P)$.*

For Example 4.7 the term whose tree is in Figure 1 is reduced by partitioning to $\lambda y.f(\lambda x z_1 z_2.x(z_2))a$. Similarly, the term in Figure 5 is reduced to $\lambda x_1 x_2.x_2(a)$.

The decidability proof exploits a key feature of game playing on a potential solution term t_0 for third-order P, the *tree-model* property: each play $\pi \in \mathsf{G}(t_0, P)$ descends a branch of t_0 until a final state is reached. Because there is only one *level* of simple non-constant tiles, so, they are both top and end, game playing is heavily constrained. With a 4th or 5th-order tree there are two levels of simple non-constant tiles: top tiles $y(\lambda\overline{x}_1, \ldots, \lambda\overline{x}_k)$ and end tiles $z(\lambda\overline{z}_1, \ldots, \lambda\overline{z}_l)$ where z is bound by a $\lambda\overline{x}_j$. The number of levels increases with order: at 8th or 9th-order there are four levels. As soon as there is more than one level, game playing may jump around the tree. To show decidability of matching for the general case, the argument is much more involved and appeals to unfolding which is analogous to unravelling in modal

logic. Unfolding induces the tree-model property. The proof of decidability uses unfolding followed by selective refolding (the inverse of unfolding) and from their combinatorial properties the small model property holds, see [26]. Here, instead we wish to consider *infinite* games on *infinite* λ-terms.

5 Higher-order program schemes

Assume $\mathbf{0}$ is the domain of finite and infinite F-trees where each node is labelled by a basis function (constant) $f \in \mathsf{F} = \{f_1, \ldots, f_k\}$. Each f_i has an *arity* which is its degree of branching: if $f_i : \mathbf{0}$ then it has arity 0 and $f : (\mathbf{0}, \ldots, \mathbf{0}, \mathbf{0})$ has arity the number of $\mathbf{0}$'s less one [2]. A higher-order program scheme, following Damm [8], is defined relative to a set of basis functions.

Definition 5.1 A *scheme* is a finite family $F_i\, x_1^i \ldots x_{n_i}^i \stackrel{\text{def}}{=} t_i$, $1 \leq i \leq m$, of definitions where each F_i is typed and distinct, and each $t_i : \mathbf{0}$ is built from the typed variables, $x_1^i, \ldots, x_{n_i}^i$, basis functions and the F_i's using application. There is also a start configuration $S : \mathbf{0}$ without free variables built from the basis functions and F_i's using application. The *order* of a scheme is the highest order of a variable x_j^i that occurs on the left hand side of a definition [3].

Example 5.2 $Fx \stackrel{\text{def}}{=} fF(g(x))g(x)$ with start configuration Fa is first-order as $x : \mathbf{0}$: here, a has arity 0, g arity 1 and f arity 2. □

Example 5.3 The following with start configuration $F\, g\, h\, a$ is second-order.

$$\begin{aligned} Fx_1x_2x_3 &\stackrel{\text{def}}{=} f\,(F(Gx_1)(Hx_2)x_3)\,x_1(x_2(x_3)) \\ Gy_1y_2 &\stackrel{\text{def}}{=} g(y_1(y_2)) \\ Hz_1z_2 &\stackrel{\text{def}}{=} h(z_1(z_2)) \end{aligned}$$

Variables $x_1, x_2, y_1, z_1 : (\mathbf{0}, \mathbf{0})$, $x_3, y_2, z_2 : \mathbf{0}$, constant a has arity 0, g, h arity 1 and f arity 2. □

A scheme is an abstracted functional program whose interpretation is the F-tree generated by S. For instance, Fa of Example 5.2 becomes $fF(ga)ga$, then $f(fF(g(g(a)))g(g(a)))g(a)$ and so on, thereby generating the infinite tree whose initial part is depicted in Figure 6. Operationally, the following transition rules generate the tree when applied to S.

(i) $F_i\, s_1 \ldots s_{n_i} \longrightarrow t_i\{s_1/x_1^i, \ldots, s_{n_i}/x_{n_i}\}$

(ii) If $s_j \longrightarrow s_j'$ for $1 \leq j \leq k$ then $f_i\, s_1 \ldots s_k \longrightarrow f_i\, s_1' \ldots s_k'$

(iii) If $a : \mathbf{0}$ then $a \longrightarrow a$

[2] The domain ordering is: $\perp \sqsubseteq \mathbf{t}$ and $\mathbf{t}_j \sqsubseteq \mathbf{t}_j'$ for each j implies $f_i\mathbf{t}_1 \ldots \mathbf{t}_k \sqsubseteq f_i\mathbf{t}_1' \ldots \mathbf{t}_k'$.

[3] The matching literature assumes $\mathbf{0}$ is of order 1, following first-order logic, whereas the scheme literature assumes $\mathbf{0}$ is of order 0 following λ-calculus literature. However, the definition of the order of a scheme is then the highest order of any F_i which coincides with the definition here when $\mathbf{0}$ has order 1.

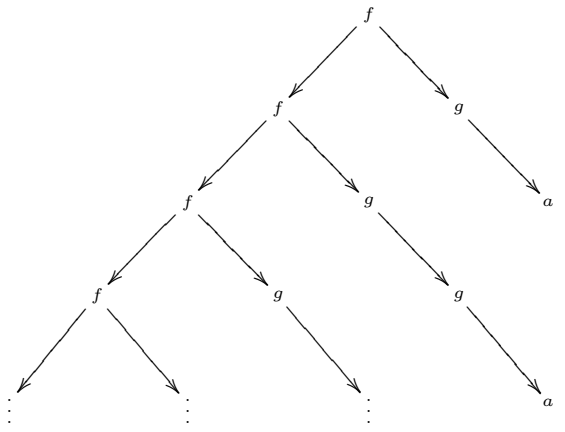


Fig. 6. The tree for Example 5.2

Denotationally, the meaning of a scheme is the least fixed point with respect to the free interpretation of basis functions, following Damm [8]. For Example 5.2, $F^0a = \perp$ and $F^{i+1}a = f(F^i(ga))ga$. So $F^2(a) = f(f(\perp)g(g(a)))g(a)$. The resulting F-tree is $F^\omega(a) = \bigsqcup_{i\geq 0} F^i(a)$ which is the meaning of $Y(\lambda F.\, \lambda x.\, f(F(gx))g(x))a$. Clearly, operational and denotational views coincide exactly. In the following we restrict attention to schemes that do not generate F-trees with undefined subtrees.

The definition of scheme, Definition 5.1, is slightly more generous than Damm's for order greater than 1. Damm's definition coincides with the *safe* schemes introduced by Knapik, Niwiński and Urzyczyn [14]: see [1,10] for a detailed discussion of this restriction and the proof that it indeed coincides with Damm's. An open question is whether safety restricts expressive power.

The classical *higher-order scheme problem* is: given two (possibly safe) schemes do they generate the same F-tree? One approach to understand this problem is to relate schemes to formal languages. Each basis function $f \in \mathsf{F}$ with arity $n > 1$ can be split into terminal symbols $f^1, \ldots, f^n$ reflecting the different directions that can be taken. The *branch language* for an F-tree is the deterministic set of words which are paths from the root to a terminal node $a : \mathbf{0}$: for Figure 6, $f^1 f^2 gga$ is such a word. Courcelle showed that the equivalence problem for first-order schemes is interreducible to the equivalence checking problem for deterministic pushdown automata [7]. This problem was subsequently positively solved by Sénizergues [21,22,24]. For $n > 1$, equivalence of *safe* nth-order recursion schemes coincides with equivalence between deterministic nth-order pushdown automata [9,14].

Indeed, an open question is whether the branch language of an F-tree generated by a higher-order scheme is (deterministic) context-sensitive. For a first-order scheme, Courcelle shows it is a deterministic context-free language. Figure 8 illustrates that expressive power reaches beyond the context-free for schemes of order more than one. The family of languages recognized by second-order pushdown automata is the *indexed languages*, introduced by Aho in 1968 [2,3], which permit some context-dependency. Aho offers a grammatical method for generating them as well as an automata theoretic method (using nested stack automata) which turns out to be equivalent to the second-order pushdown automata, as shown by Maslov

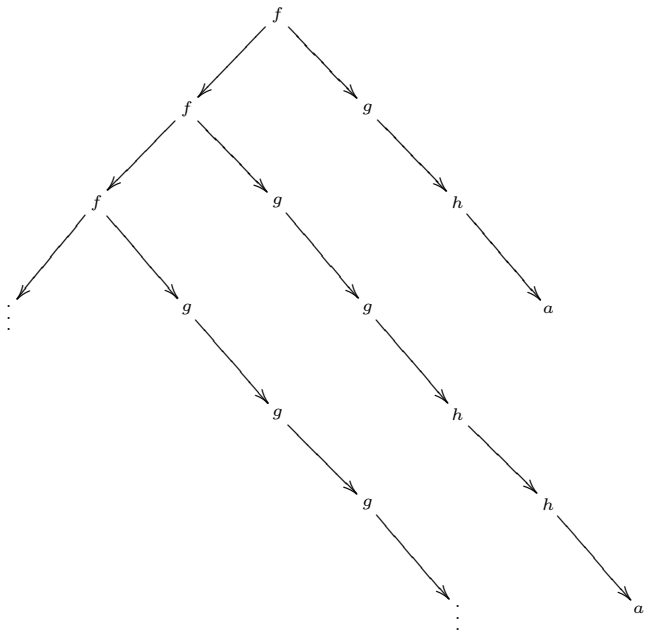


Fig. 7. The tree for Example 5.3

[16] who also defined a hierarchy of higher-order indexed languages characterized by higher-order pushdown automata, [16]. A more detailed account is given by Damm and Goerdt [9]. Aho also shows that the indexed languages are context-sensitive. They form an AFL and are a proper subset of the context-sensitive languages. It therefore follows that the branch languages of the F-trees generated by second-order *safe* schemes are deterministic context-sensitive. Aelig, de Miranda and Ong show that the branch language for an arbitrary second-order scheme is context-sensitive [1]. However, their proof simulates such a scheme using nondeterministic second-order pushdown automata: so, it is an open question whether the language is deterministic.

One area where there has been recent success in developing decision procedures for schemes is the model checking problem: given a scheme, does its F-tree have a decidable monadic second-order theory? Knapik, Niwiński and Urzyczyn proved decidability of model checking for safe schemes by transforming them into infinite terms of typed λ-calculus [14]. Ong has extended this result to all schemes. His proof also transforms a scheme into an infinite (regular) term of typed λ-calculus [17]. He then appeals to innocent game semantics to understand how it generates an F-tree (and how an alternating parity automaton that runs on its tree can be transformed into one that runs on the term instead). This provides a third account of the semantics of schemes using games. Our intention is to reformulate the game-theoretic interpretation using a similar game to that for matching.

6 Games for schemes

Knapik, Niwiński and Urzyczyn transform (safe) schemes into infinite terms of typed λ-calculus [14]. Here, we follow Ong's construction of the *long transform* of a scheme

into typed λ-calculus [17] except that we delay the introduction of dummy λ's until its tree representation (as we did in Section 3). For each definition $Fx_1 \dots x_n \stackrel{\text{def}}{=} t$ or start configuration S in a scheme we do the following.

1. Expand t or S to its η-long form (as in Section 2). One hereditarily η-expands every subterm if it occurs in an operand position (that is, as the second argument of the implicit application operator of λ-calculus).
2. Insert formal apply symbols $@_A$. Replace each ground type subterm of t and S of the form $F_i e_1 \dots e_n$ where $F_i : A = ((A_1, \dots, A_n, \mathbf{0}), A_1, \dots, A_n, \mathbf{0})$ by $@_A F_i e_1 \dots e_n$.
3. Lambda abstract the definition. If $Fx_1 \dots x_n \stackrel{\text{def}}{=} t'$ after stage 2 and $n > 0$, then replace it with $F \stackrel{\text{def}}{=} \lambda x_1 \dots x_n.t'$.
4. Rename bound variables so the set of equations and the start configuration are well named with respect to each other.

Wherever possible we omit the type A from @.

Example 6.1 Stage 1 does not apply to Example 5.2. For stage 2, the start configuration becomes $@Fa$ and the single equation becomes $Fx \stackrel{\text{def}}{=} f(@F(g(x)))g(x)$. After λ-abstraction, $F \stackrel{\text{def}}{=} \lambda x.f(@Fg(x))g(x)$. □

Example 6.2 Example 5.3 is transformed into $@F(\lambda u.g(u))(\lambda w.h(w))a$ as start configuration, where the constants g and h are η-expanded, with the following equations.

$$F \stackrel{\text{def}}{=} \lambda x_1 x_2 x_3.f(@F(\lambda x.@G(\lambda v.x_1(v))x)(\lambda y.@H(\lambda s.x_2(s))y)x_3)\,(x_1(x_2(x_3)))$$
$$G \stackrel{\text{def}}{=} \lambda y_1 y_2.g(y_1(y_2))$$
$$H \stackrel{\text{def}}{=} \lambda z_1 z_2.h(z_1(z_2))$$

In the first equation, the operand subterms Gx_1 and Hx_2 are η-expanded. □

This reformulation does not affect the tree generated using β-reduction and replacement of the F_i's by their definitions: for instance $@Fa = @(\lambda x.f(@Fg(x))(g(x)))a$ which β-reduces to $f(@Fg(a))(g(a))$ assuming the standard interpretation of @.

The next step is to represent a scheme as a tree with backedges by systematically replacing all the F_i's with their definitions. The idea is a simple generalization of the representation of terms in Section 3. First, we treat $@_A$ when $A = ((A_1, \dots, A_n, \mathbf{0}), A_1, \dots, A_n, \mathbf{0})$ as a constant with arity $n + 1$. To represent $uv_1 \dots v_k$ when u is a variable or a constant (which now includes @), we assume that a dummy λ with the empty sequence of variables is placed before any subterm $v_i : \mathbf{0}$. Moreover, any F_i is replaced with its definition. We use the notation $u \downarrow_i t'$ to represent that tree t' is the ith successor of the node u as in Section 3.

For Example 6.1 we start with the initial term tree in Figure 8 where a dummy λ is introduced above a. This is then expanded to the second term tree in the figure, again with dummy λ's. We could continue by replacing F with the subtree

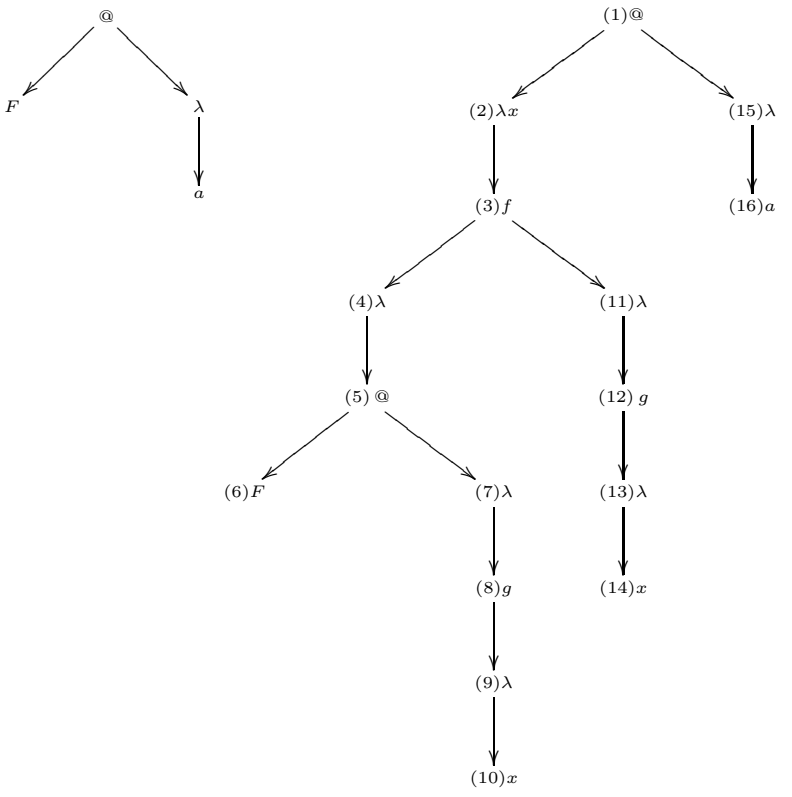


Fig. 8. The term tree for Example 6.1

rooted at (2) and so on, to produce an infinite λ-term (which only contains finitely many variables). Instead, we assume that there is an edge from (5) to (2): that is (5) $\downarrow_1$ (2). So the scheme produces a tree with backedges.

The aim is to understand the way a term generates an F-tree using games. Ong provides such a foundation using game semantics [17]. Again, to avoid justification pointers, the key is an appeal to iteratively defined look-up tables. However, the situation is simpler than for matching as there are only free variables in the tree. We appeal to single look-up tables θ_k: θ_k is a partial map from variables in t to pairs $t'\theta_j$ where t' is a node of the tree and $j < k$.

The basic game $\mathsf{G}(S)$ is played by one participant, player $\forall$. A *play* of $\mathsf{G}(S)$ is an infinite sequence of positions $t_1\theta_1, \ldots$ or a finite sequence $t_1\theta_1, \ldots, t_n\theta_n$ where $t_n = a : \mathbf{0}$. For the initial position, t_1 is the root of the tree representation of S and θ_1 is $\emptyset$, the initial empty look-up table. If the current position is $t_m\theta_m$ and t_m is not a constant $a : \mathbf{0}$ then the next position $t_{m+1}\theta_{m+1}$ is determined by a unique move in Figure 9. If play is at $\lambda\overline{y}$ then it descends the tree. At @, play proceeds to the first successor $\lambda x_1 \ldots x_n$ and the look-up table is updated accordingly (the other successors t_{i+1} with the curent look-up table is associated with x_i). At a constant f with arity more than 0, $\forall$ chooses a successor. If play is at a variable $y : \mathbf{0}$ and the entry in the current look-up table is $t'\theta_k$ then play jumps to t' and θ_k becomes the look-up table. If y is higher-order and its entry is $t'\theta_k$ in the current look-up table θ_m where $t' = \lambda z_1 \ldots z_n$ then play jumps to t' and the look-up table is θ_k together with the association of $t'_i\theta_m$ for each z_i when $y \downarrow_i t'_i$.

A. $t_m = \lambda\overline{y}$ and $t_m \downarrow_1 t'$. Then $t_{m+1} = t'$ and $\theta_{m+1} = \theta_m$.

B. $t_m = @_A$, $A = ((A_1, \ldots, A_n, \mathbf{0}), A_1, \ldots, A_n, \mathbf{0})$, $t_m \downarrow_i t'_i$ for $1 \leq i \leq n+1$ and $t'_1 = \lambda x_1 \ldots x_n$. Then $t_{m+1} = t'_1$ and $\theta_{m+1} = \theta_m\{t'_2\theta_m/x_1, \ldots, t'_{n+1}\theta_m/x_n\}$.

C. $t_m = f$, f has arity $n > 0$ and $t_m \downarrow_i t'_i$ for $1 \leq i \leq n$. Then $\forall$ chooses a direction $j : 1 \leq j \leq n$ and $t_{m+1} = t'_j$ and $\theta_{m+1} = \theta_m$.

D. $t_m = y : \mathbf{0}$ and $\theta_m(y) = t'\theta_k$. Then $t_{m+1} = t'$ and $\theta_{m+1} = \theta_k$.

E. $t_m = y : (A_1, \ldots, A_n, \mathbf{0})$, $t_m \downarrow_i t'_i$ for $1 \leq i \leq n$ and $\theta_m(y) = t'\theta_k$ where $t' = \lambda z_1 \ldots z_n$. Then $t_{m+1} = t'$ and $\theta_{m+1} = \theta_k\{t'_1\theta_m/z_1, \ldots, t'_n\theta_m/z_n\}$

Fig. 9. Game moves

Example 6.3 Consider $\mathsf{G}(S)$ when S is the tree in Figure 8. The first moves are as follows.

$$(1)\theta_1 \quad (2)\theta_2 = \theta_1\{(15)\theta_1/x\} \quad (3)\theta_2$$

At this point there is a $\forall$-choice. If play takes the right branch then play continues.

$$(11)\theta_2 \quad (12)\theta_2 \quad (13)\theta_2 \quad (14)\theta_2$$

Now x is called: $\theta_2(x) = (15)\theta_1$. So play continues with a jump: $(15)\theta_1$ and ends at $(16)\theta_1$. The branch associated with this play is f^2ga. If the other choice is taken at position (3) then it continues.

$$(4)\theta_2 \quad (5)\theta_2 \quad (2)\theta_3 = \theta_2\{(7)\theta_2/x\} \quad (3)\theta_3$$

And again there is a $\forall$-choice. If the right branch is taken then play continues.

$$(11)\theta_3 \quad (12)\theta_3 \quad (13)\theta_3 \quad (14)\theta_3$$

Again x is called: $\theta_3(x) = (7)\theta_2$. So play now jumps to (7) and proceeds to (10) before jumping to (15) and finishing at (16): the branch associated with this play is f^1f^2gga. There is just one infinite play in $\mathsf{G}(S)$ when the left branch at (3) is always chosen. □

Example 6.4 We now examine plays of $\mathsf{G}(S)$ for S in Figure 10. Play starts as follows.

$$(1)\theta_1 \quad (2)\theta_2 = \theta_1\{(42)\theta_1/x_1, (46)\theta_1/x_2, (50)\theta_1/x_3\} \quad (3)\theta_2$$

If $\forall$ chooses the right branch of f then play continues.

$$(36)\theta_2 \quad (37)\theta_2 \quad (42)\theta_3 = \theta_1\{(38)\theta_2/u\}$$

Because $\theta_2(x_1) = (42)\theta_1$, when play jumps to (42) the look-up table associates $(38)\theta_2$ with u. Play proceeds to (45) and returns to (38).

$$(43)\theta_3 \quad (44)\theta_3 \quad (45)\theta_3 \quad (38)\theta_2 \quad (39)\theta_2$$

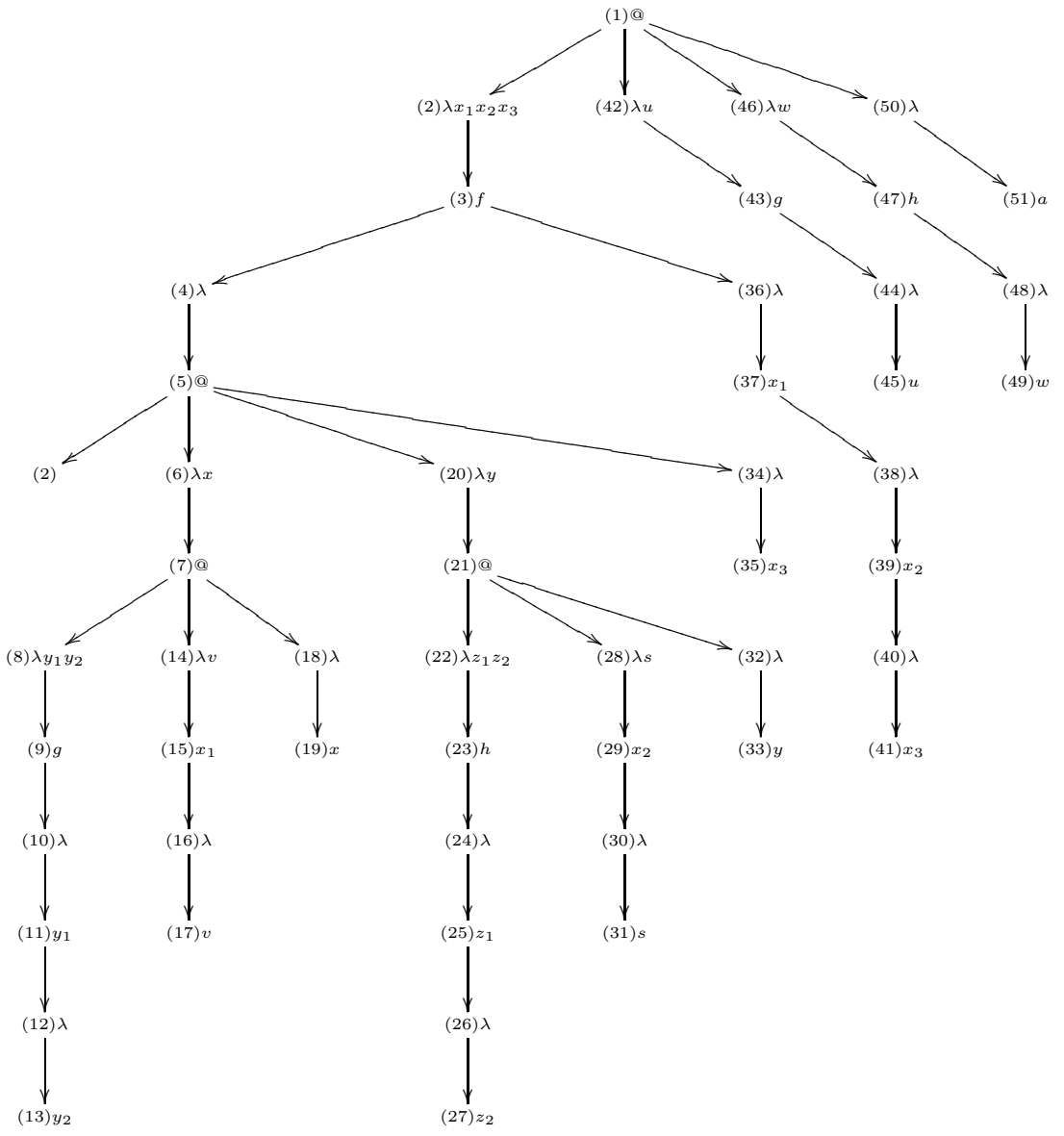


Fig. 10. The term tree for Example 6.2

Now it jumps to (46)

$$(46)\theta_4 = \theta_1\{(40)\theta_2/w\} \quad (47)\theta_4 \quad (48)\theta_4 \quad (49)\theta_4$$

and returns to (40).

$$(40)\theta_2 \quad (41)\theta_2 \quad (50)\theta_1 \quad (51)\theta_1$$

The branch associated with this play is f^2ghha. □

Given a scheme S, the F-tree *generated from the game* $\mathsf{G}(S)$ is the tree with internal nodes f and subtrees determined by ∀'s choice when position C of Figure 9 is encountered, and leaf nodes a when play finishes at this constant.

Proposition 6.5 *The* F*-tree generated by* S *is the* F*-tree generated by* $\mathsf{G}(S)$.

Proof. We show that each prefix of the F-tree generated by S is the same prefix of the F-tree generated by $\mathsf{G}(S)$ and that the remaining subtrees are the same. To do this we show that the operational semantics of a scheme S that is expanded into normal form when look-up tables θ_i are interpreted as delayed substitutions is isomorphic to game playing. Initially, the scheme operationally is S and the intial move in $\mathsf{G}(S)$ is $S\theta_1$ where θ_1 is the empty look-up table: consequently the same. Assume current branch $f_1^{i_1} \dots f_k^{i_k}$ in the prefix of the F-tree and $t_m\theta_m$ is the current position in the play. We proceed by cases on t_m. If $t_m = @$, so $@(\lambda x_1 \dots x_n.t')t_2 \dots t_{n+1}\theta_m$ is the current unfolding in the operational semantics: this generates the F-tree $t'\theta_m\{t_2\theta_m/x_1, \dots, t_{n+1}\theta_m\}$. Game theoretically, play first goes to the node labelled $\lambda x_1 \dots x_n$ with $\theta_{m+1} = \theta_m\{t_2\theta_m/x_1, \dots, t_{n+1}\theta_m\}$ and then play goes to $t'\theta_{m+1}$, as required. If $t_m = f$ and so $ft_1 \dots t_n\theta_m$ is the current unfolding in the operational semantics then the branch $f_1^{i_1} \dots f_k^{i_k}$ is expanded with f^j for $1 \leq j \leq n$, and the jth subtree is $t_j\theta_m$. Clearly, in the game if $\forall$ chooses direction j then the next position is $t_j\theta_m$. If $f : \mathbf{0}$ then it is a leaf of the F-tree and $t_m\theta_m$ is a final position in a play. If $t_m = y$ and so $yt_1 \dots t_n\theta_m$ is the current unfolding in the operational semantics then this generates the F-tree $\theta_m(y)t_1\theta_m \dots t_n\theta_m$: if $\theta_m(y) = \lambda z_1 \dots z_n.t'\theta_k$ then this reduces to $t'\theta_k\{t_1\theta_m/z_1, \dots, t_n\theta_m/z_n\}$ and the game follows suit. A similar argument holds for the case $y : \mathbf{0}$. □

The issue is whether the game-theoretic characterization of the F-tree generated by a scheme can underpin useful decision procedures. Ong [17], using game-semantics, shows how an alternating parity automaton that runs on the F-tree can be transformed into one that runs on the term tree. The idea is to include assumptions about play jumping with the descent of an automaton down a leftmost successor of @: these assumptions are then checked by spawning auxiliary automata that descend the other successors of the @. We can employ the tile classification of regions of the scheme tree, and their subplays as described for matching, as an alternative basis for this decidability proof. Much more work is needed to discover relationships between the static structure of a scheme and the dynamics of game playing.

References

[1] Aelig, K., de Miranda, J. and Ong, C-H. L. (2005). Safety is not a restriction at level 2 for string languages. *Lecture Notes in Computer Science*, **3411**, 490-501.

[2] Aho, A. (1968). Indexed grammars–an extension of context-free grammars. *Journal of ACM*, **15**, 647-671.

[3] Aho, A. (1969). Nested stack automata. *Journal of ACM*, **16**, 383-406.

[4] Baeten, J., Bergstra, J., and Klop, J. (1993). Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of ACM*, **40**, 653-682.

[5] Caucal, D. (2002). On infinite terms having a decidable monadic theory. *Lecture Notes in Computer Science*, **2420**, 165-176.

[6] Comon, H. and Jurski, Y. Higher-order matching and tree automata. *Lecture Notes in Computer Science*, **1414**, 157-176, (1997).

[7] Courcelle, B. (1978). A representation of trees by languages I and II. *Theoretical Computer Science*, **6**, 255-279 and **7**, 25-55.

[8] Damm, W. (1982). The IO- and OI-hierarchy. *Theoretical Computer Science*, **25**, 95-169.

[9] Damm, W., and Goerdt, A. (1986). An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, **71**, 1-32.

[10] De Miranda, J. (2006). *Structures Generated by Higher-Order Grammars and the Safety Constraint*, D.Phil thesis submitted, University of Oxford.

[11] Dowek, G. Third-order matching is decidable. *Annals of Pure and Applied Logic*, **69**, 135-155, (1994).

[12] Huet, G. *Rèsolution d'èquations dans les langages d'ordre* 1, 2, ... ω. Thèse de doctorat d'ètat, Universitè Paris VII, (1976).

[13] Joly, T. Encoding of the halting problem into the monster type and applications. *Lecture Notes in Computer Science*, **2701**, 153-166, (2003).

[14] Knapik, T., Niwiński, D., and Urzyczyn, P. (2002). Higher-order pushdown trees are easy. *Lecture Notes in Computer Science*, **2303**, 205-222.

[15] Loader, R. Higher-order β-matching is undecidable, *Logic Journal of the IGPL*, **11(1)**, 51-68, (2003).

[16] Maslov, A. (1976). Multilevel stack automata. *Problems of Information Transmission*, **12**, 38-43.

[17] Ong, C.-H. L. *On model-checking trees generated by higher-order recursion schemes*, Procs LICS 2006, 81-90. (Longer version available from Ong's web page, 42 pages preprint.)

[18] Padovani, V. Decidability of all minimal models. *Lecture Notes in Computer Science*, **1158**, 201-215, (1996).

[19] Padovani, V. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, **10(3)**, 361-372, (2001).

[20] Schubert, A. Linear interpolation for the higher-order matching problem. *Lecture Notes in Computer Science*, **1214**, 441-452, (1997).

[21] Sénizergues, G. (2001). L(A) = L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, **251**, 1-166.

[22] Sénizergues, G. (2002). L(A) = L(B)? a simplified decidability proof. *Theoretical Computer Science*, **281**, 555-608.

[23] Stirling, C. *Modal and Temporal Properties of Processes*, Texts in Computer Science, Springer, (2001).

[24] Stirling, C. (2002) Deciding DPDA equivalence is primitive recursive. *Lecture Notes in Computer Science*, **2380**, 821-832.

[25] Stirling, C. Higher-order matching and games. *Lecture Notes in Computer Science*, **3634**, 119-134, (2005).

[26] Stirling, C. *A game-theoretic approach to deciding higher-order matching*, *Lecture Notes in Computer Science*, **4052**, 348-359, (2006).